The Generalized Support Software (GSS)

A Description of Its Current Software Development Process

Prepared by
the Software Engineering Laboratory

February 12, 1996

Table of Contents

1. Purpose 1

2. Overview 2

2.1 Background 2

2.2 GSS Architecture 2

2.3 GSS Process 3

3. Domain Analysis Phase 8

3.1 Preliminary Work 8

3.2 Initial Domain Analysis Phase 9

3.3 Sustaining Domain Analysis Phase 10

3.4 Further Iterations of Domain Analysis 10

4. Component Engineering Phase 12

4.1 Preliminary Work 12

4.2 Initial Component Engineering Phase 12

4.3 Sustaining Component Engineering Phase 13

4.4 Further Iterations of Component Engineering 13

5. Mission Analysis Phase 15

6. Application Configuration Phase 17

7. Application Test Phase 20

Figures

Figure 1. The GSS Architecture Hierarchy 4

Figure 2. GSS Architecture Example 4

Figure 3. GSS Component Development and Application Deployment Process 5

Figure 4. Overlapping Phases From Domain Analysis Through Application Testing 6

Figure 5. Domain Analysis Phase Activities and Products 9

Figure 6. Progress of the Domain Analysis Team Through the Overall Domain 11

Figure 7. Component Engineering Phase Activities and Products 14

Figure 8. Mission Analysis Phase Activities and Products 16

Figure 9. Application Configuration Phase Activities and Products 19

Figure 10. Application Testing Phase Activities and Products 20

1. Purpose

This paper documents the software development process used by the Generalized Support Software (GSS) project. The purpose of the paper is to capture the current process in order to subsequently evaluate it and recommend improvements to it.

To date the GSS process has been used to develop components for the Advanced Attitude System (AAS) and, more recently, for the Automated Mission Planning Tool (AMPT). The AAS has been implemented using Ada 83, but there are plans to convert it to C++. The AMPT is being implemented in C++. Except for language-dependent differences, the AMPT is following largely the same development process as is the AAS. This paper refers to their common development process as "the GSS process."

2. Overview

2.1 Background

The Flight Dynamics Division (FDD) of NASA's Goddard Space Flight Center develops and maintains software applications in order to support spacecraft missions. These applications fall into three broad areas: spacecraft attitude support software, spacecraft orbit support software, and mission planning software. Although each new mission usually requires the same kinds of applications as previous missions, each mission also has unique hardware and mission requirements. Because of this, each new mission has usually required tailor-made applications which are similar in overall design and functionality to previous applications but are different in detail.

In the past decade when the FDD has built new applications it has attempted to reuse from previous applications as many software modules as possible, with as few modifications as possible. To a large extent the FDD has been successful in achieving high reuse. In order to further increase the amount (and the type) of reuse, and at the same time drastically reduce the cycle time needed to field each new application, the FDD has embarked on a new process in the last two years. This process was first utilized for applications dealing with spacecraft attitude support (AAS) and recently was expanded to include applications dealing with mission planning (AMPT).

The overall goal of the GSS is to develop a strategic reuse asset library that enables much more rapid deployment of flight dynamics applications. The GSS development process achieves this rapid deployment by utilizing an object-oriented architecture in which the reusable assets are the generalized *specifications* for the reusable software components, as well as the reusable software components themselves, called *classes*. Adopting this architecture and process results in a paradigm shift from developing software applications to *configuring* software applications. These three key facets of GSS are:

- an architecture for rapid deployment of Flight Dynamics applications
- a strategic reuse library for Flight Dynamics classes
- a paradigm shift from developing software to configuring software for mission support

2.2 GSS Architecture

Whereas classes (and their specifications) are the key elements of the reuse asset library, the key elements of a configured software application are the *objects* that compose the application. An object is a specific instance of a class (called an *instantiation* in object-oriented design terminology). By way of analogy, a "size-12 brown Oxford" can be considered an instance of the class "shoe." A class has certain characteristics that define it and distinguish it from other classes, just as all shoes share the general characteristics of size, color, and style. In object-oriented programming the class acts like a template. When the programmer specifies a value for each of the class's characteristics (size = 12, color = brown, style = Oxford), the program instantiates the desired object from the class.

In the Flight Dynamics domain, an object is a model of some individual item of interest in that domain. The object includes both the *data parameters* that describe the object and the *functions* that specify appropriate processing. Below are several examples of Flight Dynamics classes. From each of these classes one could instantiate several different objects.

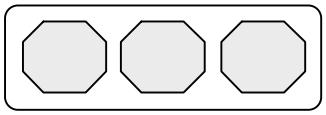
- three-axis stabilized attitude model
- axisymmetric nutating attitude model
- orbit model
- Sun-pointing coordinate system model
- V-slit Sun sensor model
- high-gain antenna model
- Moon model
- dipole geomagnetic field model
- star identification double match model
- quaternion estimator model

In addition to objects and classes, the GSS architecture hierarchy consists of *categories,* and *subdomains.* All four terms are defined in Figure 1. Figure 2 presents the GSS Hardware Subdomain as an example of this architecture. It demonstrates how the Hardware Subdomain is composed of two categories, Sensor Models and Actuator Models. It then lists all six actuator classes in the Actuator Model Category. The sensor classes are too numerous to include all in the figure.

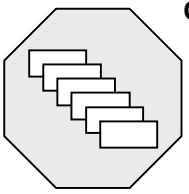
2.3 GSS Process

The GSS process, depicted in Figure 3, consists of two overlapping subprocesses involving five different teams. In the Component Development Subprocess the Domain Analysis Team analyzes the FDD domain, defines the classes, categories, and subdomains, and writes the class and category specifications, which become part of the reusable asset library. As these specifications are created, the Component Engineering Team takes the specifications and develops the code for these classes and categories. These coded classes and categories also become part of the reusable asset library.

In the Application Deployment Subprocess three other teams make use of the specifications and code from the reusable asset library in order to configure, test, and deliver FDD software applications. As soon as some of the specifications are written by the domain analysts, the Mission Analysis Team begins mapping these specifications against the new mission's requirements. In this way the team determines which classes and categories are needed for the new mission. The Mission Analysis Team also defines the parameters of the specific objects which must be instantiated from the classes. They pass these *application and object specifications* along to the Application Configuration Team, which then configures the application from the classes and categories in the reusable asset library. Finally the Application Test Team tests the new application and certifies it for delivery to mission support for operational use.



Subdomain: a group that contains all categories necessary to specify the functionality in a specific high-level area of the overall problem domain.



Category: a set of similar classes grouped together along with rules for using member classes for mission support.

- every class must belong to a category
- a category might have only one class
- a category specifies a common functional interface for its member classes
- each class must specify *at least* all functions specified for its category
- a class may also specify additional functions unique to that class

Class: a group of objects specified together

- all objects in a class share the same specifications of data and functionality
- a class specification is effectively a "template" for individual objects in the class

Reuse Library

Applications

Object: a model of some individual item of interest in the problem domain, including both

- *data parameters* that describe the object
- *functions* that specify appropriate processing

Figure 1. The GSS Architecture Hierarchy

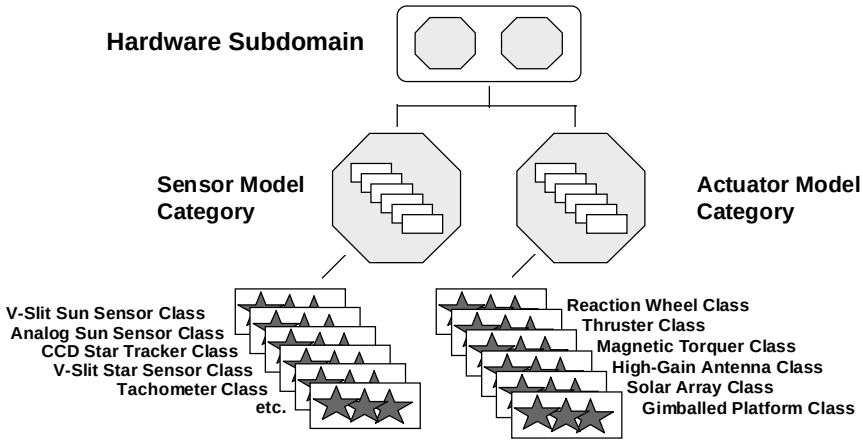


Figure 2. GSS Architecture Example

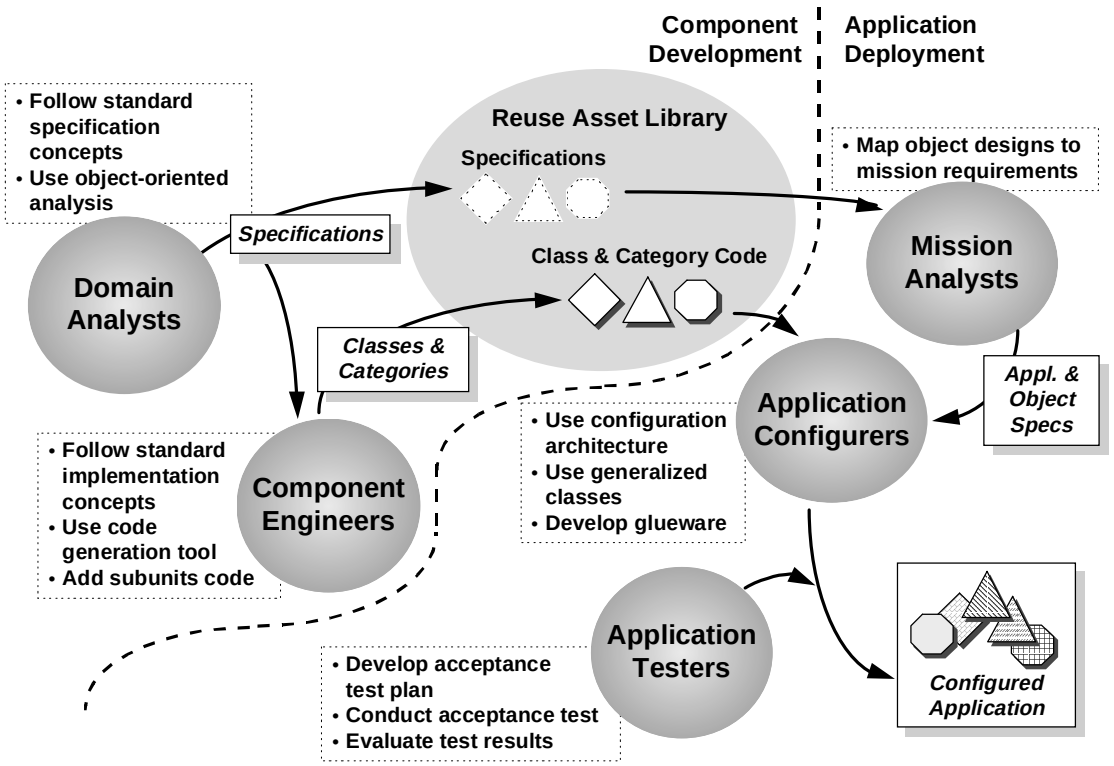


Figure 3. GSS Component Development and Application Deployment Process

Each subprocess can be thought of as consisting of several overlapping phases, one for each of the major teams in the subprocess, as follows.

1. Component Development Subprocess
    - 1.1. Domain Analysis Phase
    - 1.2. Component Engineering Phase
2. Application Deployment Subprocess
    - 2.1. Mission Analysis Phase
    - 2.2. Application Configuration Phase
    - 2.3. Application Testing Phase

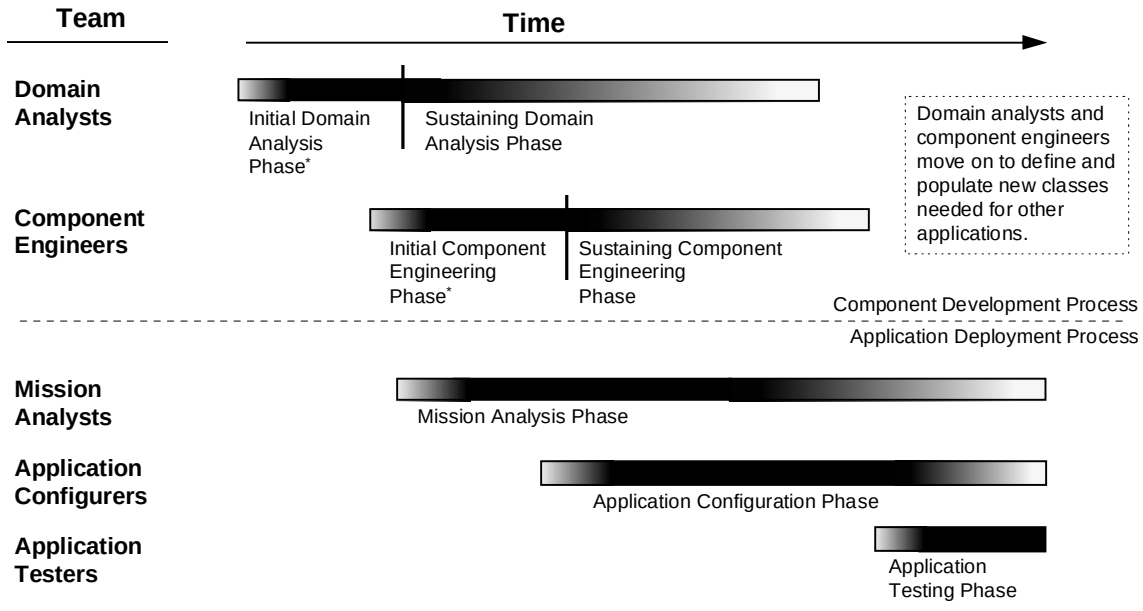
Figure 4 portrays how these five phases overlap in time during the populating of the reuse asset library and the deployment of the *first* application of a given type, such as the first telemetry simulator. The darker the team bar, the heavier the involvement of the team at that time. Note that the Domain Analysis Phase and the Component Engineering Phase are each subdivided into two subphases, an initial phase and a sustaining phase. During the initial phase each team populates the reuse asset library with the stock of items (specifications from the analysts, code from the engineers) believed to be necessary and sufficient to configure the given application. During the respective sustaining phases, the

analysts and engineers modify or add to these items in the asset library when new capabilities are specified for existing assets or in response to requests from other teams.

The goal of the domain analysts and component engineers is to populate the reuse asset library with specifications and code for all the classes and categories required to define the Flight Dynamics domain. Once this is accomplished, the three teams doing application deployment can deploy application after application for mission after mission, with only modest assistance from the domain analysts and component engineers in their "sustaining" roles.

In order to benefit from the reuse asset library *while it is under construction*, the work of the two teams doing component development is coordinated with the work of the three teams doing application deployment in the manner shown in Figure 4. That is, the domain analysts and component engineers focus first on creating the classes and categories that are required to complete the first application type that is targeted, for example a telemetry simulator. Once the classes and categories for the telemetry simulator domain are created, the three application deployment teams are ready to configure one telemetry simulator after another (for one mission after another) with little help from the domain analysts and component engineers. Therefore, as the domain analysts and component engineers shift into "sustaining" the classes and categories of the telemetry simulator domain, they simultaneously begin to focus their energies on creating the classes and categories necessary to configure the next type of target application, such as a real-time attitude determination system (RTADS).

When it comes time to configure another telemetry simulator, for a subsequent mission, the Initial Domain Analysis Phase and the Initial Component Engineering Phase are not



* The Initial Domain Analysis Phase and the Initial Component Engineering Phase are executed only for the *first* example of each type of application, such as the first telemetry simulator, the first RTADS.

Figure 4. Overlapping Phases from Domain Analysis through Application Testing

repeated. These phases have already defined and coded the majority of classes and categories necessary for any telemetry simulator. New mission requirements for the next telemetry simulator might require modifications to these existing classes and categories or might possibly require adding a few new categories or classes. These are done in the sustaining subphases.

Each phase is discussed separately in the following sections of this paper. The major activities and products for each of these phases are summarized in the accompanying figures.

3. Domain Analysis Phase

The activities performed during the Domain Analysis phase correspond roughly to the activities defined for the Requirements Analysis, Preliminary Design, and Detailed Design phases of the Software Engineering Laboratory (SEL) Recommended Approach. (This phase builds on the Requirements Definition activities of past and present missions, as described below. The Requirements Definition activities for *future missions* are performed by the mission analysts during the mission analysis phase, described in a later section.) The primary difference between GSS and the Recommended Approach is that the scope of the Domain Analysis work focuses on an entire domain (family of related systems for multiple missions) instead of on a single mission system.

Because the domain is so large, however, the domain analysis team follows an iterative process. In the first iteration they define in detail the classes and categories needed for a telemetry simulator (while keeping in mind as much as possible that many of these classes and categories will also be used by other applications). In each subsequent iteration, the team then expands its focus to include all additional classes and categories required for one or more other application types. Each expansion sometimes exposes the need to redefine some previously defined classes and categories. In this way the team gradually approaches its goal of defining the entire FDD domain.

3.1 Preliminary Work

The domain analysis team begins by developing an initial high level model of the entire attitude support problem domain. Then the team develops several iterations of each of several partial subdomains within this overall domain. This exercise allows the team to evolve and refine the standards for documenting the domain architecture (design) and the standards for developing the detailed specifications for the categories and classes. These standards are documented in the GSS Specification Concepts document.

At this point the team is ready to begin applying these standards by writing the detailed specifications for that segment of the problem domain required for the first chosen application type (a telemetry simulator), keeping in mind the particular needs of the first mission that will configure one of these applications (TRMM). The domain analysis work is divided into two subphases, an initial domain analysis phase and a sustaining domain analysis phase. See Figure 5.

3.2 Initial Domain Analysis Phase

During the initial domain analysis phase, the domain analysts review requirements for telemetry simulators in the domain of past, present, and (when possible) future missions to define the superset of functionality that bounds this segment of the problem domain. Afterwards they begin creating an architecture (design) for the subdomains, categories, and classes in the telemetry simulator domain. This architecture can be viewed as the

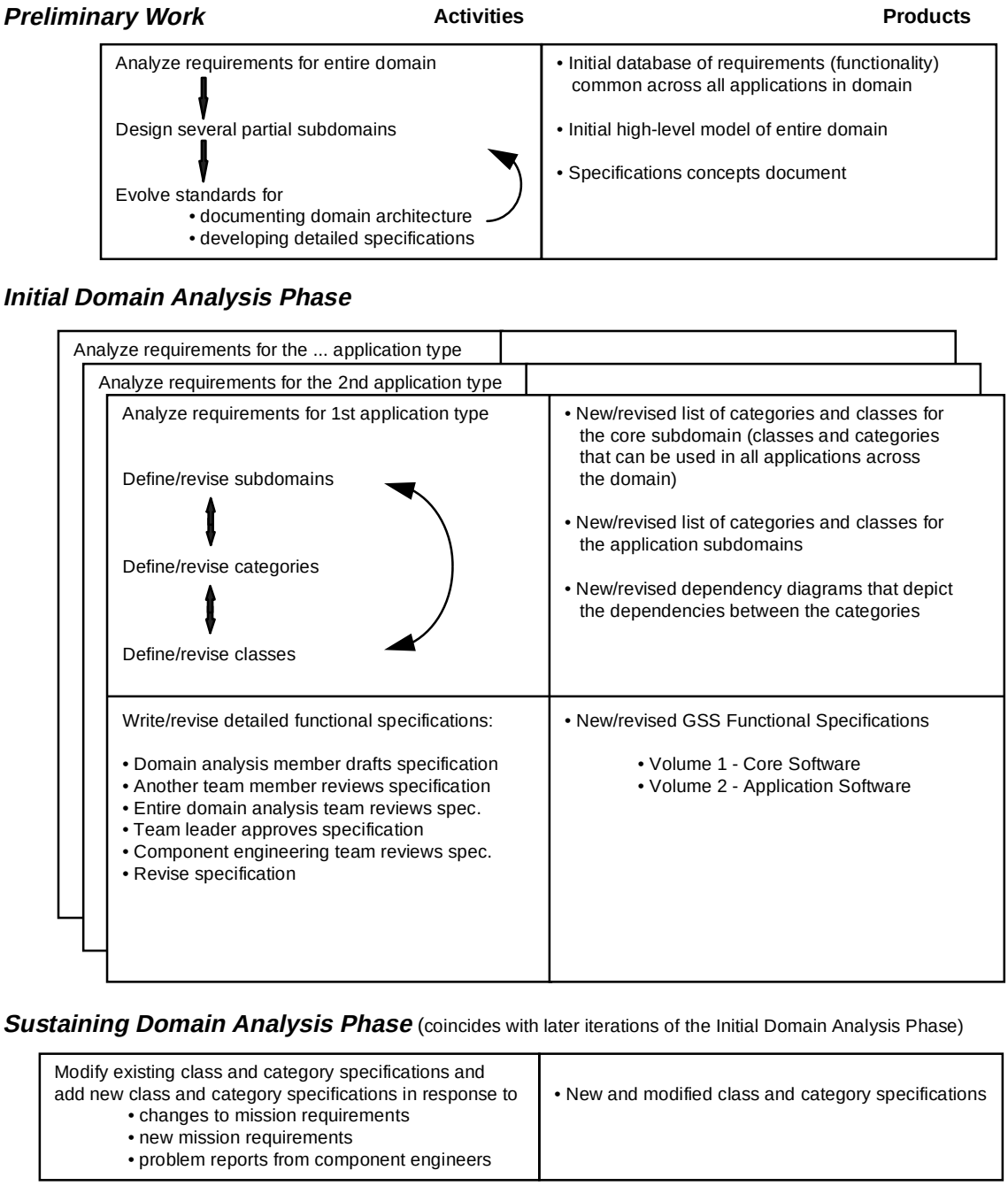


Figure 5. Domain Analysis Phase Activities and Products

preliminary design for the telemetry simulators to be configured later.

The domain analysts also assign responsibilities (member functions and data) to each class and define with which other classes each class needs to collaborate (depending on the services provided by the class). Ultimately, the domain analysts write the detailed functional specifications for each class and category. These detailed specifications can be viewed as the detailed design for the telemetry simulators.

The method adopted by the domain analysts for developing the design is a flexible, iterative process that combines *both* bottom-up and top-down approaches. For example, sometimes a set of classes is defined first and then is divided into subdomains. Sometimes a subdomain is defined first and then is partitioned into categories and classes. In some iterations classes might be transferred from one subdomain to another, or an already defined subdomain might be split into two or more subdomains.

In order to ensure their accuracy, the detailed specifications pass through a thorough review process. First each specification is written by one member of the domain analysis team, reviewed by another member of the team, then reviewed by the whole team, and finally approved by the team leader. Then the component engineering team reviews the specification and submits change requests to the domain analysts team. The domain analysts review the change requests and make modifications where necessary.

The initial domain analysis phase ends when the domain analysts have completed specifying all categories and classes that they believe are necessary to produce a telemetry simulator.

3.3 Sustaining Domain Analysis Phase

During the sustaining domain analysis phase, domain analysts develop specification modifications for the category and class functional specifications based on new mission requirements and changes to existing mission requirements. They also develop specification modifications in response to problem reports received from the component engineering team. This work includes modifying existing class and category specifications and adding new class and category specifications.

3.4 Further Iterations of Domain Analysis

After completing the initial domain analysis phase for the telemetry simulator domain, and while continuing to carry out the sustaining domain analysis phase, the domain analysis team begins the initial domain analysis phase for the RTADS domain. The team writes specifications for many new classes and categories that are required for RTADS but were not required for telemetry simulators. Where the RTADS domain overlaps the telemetry simulator domain, some rework is required. For example the hardware subdomain is split into two categories, one for sensor models and one for actuator models.

When the initial domain analysis phase for the RTADS domain is well along, the domain analysis team begins the initial domain analysis phase for the Attitude Determination System (ADS) domain, which shares much functionality with the RTADS domain. This iteration requires few new classes and less rework than the preceding iteration. Next the

team expands its focus to the mission planning domain. Figure 6 depicts these overlapping domains and the order in which the domain analysis team iterated through them.

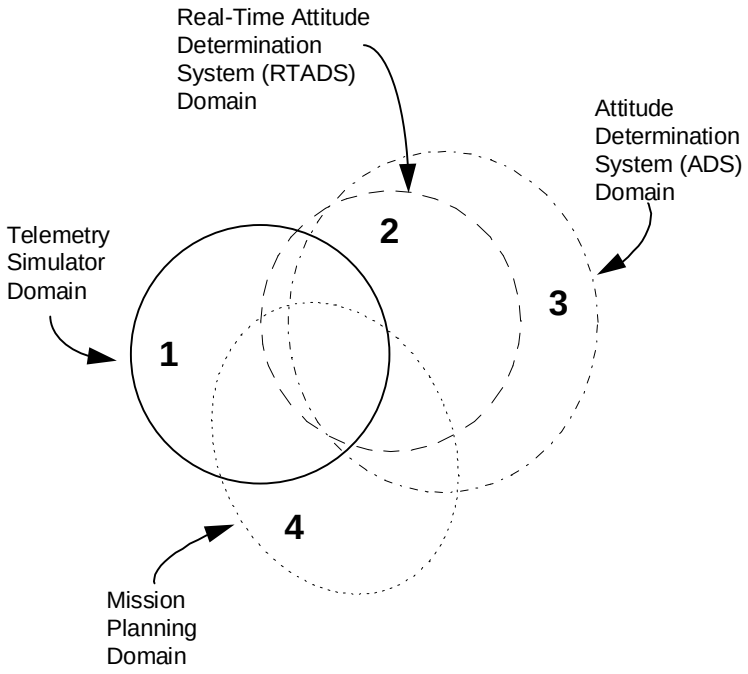


Figure 6. Progress of the Domain Analysis Team Through the Overall Domain

4. Component Engineering Phase

In the component engineering phase, the component engineers develop the code for the GSS classes and categories based on the generalized specifications. The activities performed in the component engineering phase can be thought of as roughly corresponding to some of the activities in the SEL Recommended Approach's Preliminary Design, Detailed Design, and Implementation phases.

The activities of the component engineers are closely linked to the iterative activities of the domain engineers. Consequently, the component engineers first implement the classes and categories necessary for the telemetry simulator domain, then implement any additional classes and categories necessary for the RTADS domain, ADS domain, and so on. Each expansion may require some rework on previously implemented classes and categories.

4.1 Preliminary Work

The component engineers work with the domain engineers to first define (1) the standard design for the classes and categories, (2) the standard design and behavior model for the applications programs to be configured from the classes and categories in the asset library (including a list of valid decision points and user actions), and (3) standards for developing the code for the classes and categories. These standards are documented in the GSS Implementation Concepts for each implementation language.

Since by agreement all classes share a standard design, the component engineering team produces a tool (called Classgen) to automatically generate the standard portions of each class. Once the Classgen code generator has been tested, its output doesn't need to be inspected.

The concept development and Classgen development is carried out in a long series of prototypes that develop support code and standard component designs. In order to reduce rework, the process requires that the Classgen input be a configured product, and that generated code be kept in separate files from the stubs for the user code.

Much of this architectural design work would traditionally be considered part of preliminary design. In GSS the design duties can be thought more naturally as falling under *functional architecture* (primarily the responsibility of the domain analysts in the domain analysis phase) and *implementation architecture* (primarily the responsibility of the component engineers in the component engineering phase).

4.2 Initial Component Engineering Phase

Like the domain engineering phase, the component engineering phase also consists of two subphases, initial component engineering and sustaining component engineering. In the initial component engineering phase, the component engineers work with the domain engineers to populate the reuse asset library with reusable software components (classes and categories). Based on input files written by the component engineers, the Classgen code generator generates 75% of the class code. The component engineers then write the

remaining class code (i.e., code for subunits of class packages that implement the class functionality), plus all category code. The component engineers document any deviations from the functional specifications. They also inspect code and conduct unit tests. The initial component engineering phase ends when the component engineers have completed implementing all categories and classes that are believed necessary to produce a telemetry simulator.

4.3 Sustaining Component Engineering Phase

During the sustaining component engineering phase, the component engineering team receives draft specification modifications and specifications for new classes and categories from the domain engineering team. They review the draft specification changes with the domain engineers and return the updated specifications to the domain engineers. Once the specifications have been approved by the domain engineering team, the component engineers implement the changes to the classes and categories and develop the software for the new classes and categories.

The area of unit testing sometimes contains one noteworthy difference between the initial component engineering phase process and the sustaining engineering phase process. During the initial component engineering phase, the code for the classes and categories is unit tested, and unit test plans and results are inspected. In the sustaining engineering phase, unit testing for new and/or changed classes might be waived. If unit testing is waived, the GSS process relies on code inspections—together with integration testing by the application configuration team—to find errors before application acceptance testing. The flexibility to forego unit testing follows from the experience that inspections are effective at finding errors and that unit testing is expensive. In order for inspections to be deemed sufficiently effective to waive unit testing, however, the process requires (1) a mature code generator, (2) developers with high language experience to act as inspectors, and (3) inspectors who know specification concepts and the particular specification. Items (1) and (2) are both true for code being developed in Ada83, but are not true yet for code being developed in C++.

4.4 Further Iterations of Component Engineering

After completing the initial component engineering phase for the telemetry simulator, and while continuing to carry out the sustaining component engineering phase, the component engineering team begins the initial component engineering phase for the RTADS domain. The team implements new classes and categories that are required for RTADS but were not required for telemetry simulators. Where the RTADS domain overlaps the telemetry simulator domain, some rework is required. When the team enters the sustaining component engineering phase for the RTADS domain, it also begins focusing on the initial component engineering phase for the ADS domain. In this way the team continues expanding its focus until all classes and categories for the entire Flight Dynamics domain are implemented. See Figure 7 for a summary of the activities and products for the component engineering phase.

Preliminary Work

Activities	Products
Define standard design for classes and categories Define standard design and behavior model for applications programs to be configured	• Behavior model for the application, including list of valid decision points and user actions • Implementation Description Concepts for each implementation language
Define standards for developing code for classes and categories Produce and test the Classgen code generator	• Classgen code generator and validated output

Initial Component Engineering Phase

Review class and category specifications for the ...

Review class and category specifications for the *2nd*

Review class and category specifications for the *1st* application type	• Updates to class and category specifications (delivered to domain engineering team)
Develop Classgen input file (classes only)	• Classgen input file (defines information about class data structures, member functions, dependencies, etc.)
Run Classgen code generator (classes only)	• Classgen output
Develop code for categories Develop code for subunits	• Code for categories • Code for subunits of class package that implements the required functionality of class
Prepare documentation for classes and categories	• Documented deviations from the functional specifications
Inspect code for classes and categories	• Completed inspection forms (to PAO) • Certified class code. Note: the code generated by Classgen is not inspected.
Unit test class and category code; inspect results	• Certified unit test plan • Validated code
Configure class in controlled library	• SEL COFs for new classes & categories • SEL CRFs for modified classes and categories • Configured class & category software

Sustaining Component Engineering Phase

(coincides with later iterations of the Initial Component Engineering Phase)

Modify existing class and category code and add new class and category code in response to • changes to mission requirements • new mission requirements • problem reports	• New and modified class and category code

Figure 7. Component Engineering Phase Activities and Products

5. Mission Analysis Phase

The mission analysts define the system operations concepts and requirements for the flight dynamics applications to be developed for a new spacecraft mission as described in the SEL Recommended Approach's Requirements Definition phase. These analysts produce the standard system operations concepts and system requirements specifications documents.

The mission analysis phase is the first phase in the Application Deployment Process. The mission analysis phase can begin as soon as the necessary specifications have been written by the domain analysts and while the component engineers are implementing the classes and categories.

Instead of producing the detailed functional specifications for each major application, the mission analysts work with the GSS domain analysis team to review the generalized specifications for the GSS classes and categories in order to determine which classes can be used to build the required flight dynamics applications for the mission. During this review process, they determine which classes and categories can be used verbatim, which require modifications, and which—if any—need to be written. The new classes may be generalized or "mission or application specific." If the domain analysts and mission analysts determine that a new class can be developed in generalized form, the domain analysis team adds the detailed functional specifications for the new class to the GSS Generalized Functional Specifications documents. On the other hand, if the domain analysts and the mission analysts determine that a new class is mission or application specific, then the mission analysts document the specifications for the class in the mission specification document for the application.

The results of the review of the GSS classes are documented in a mission specification document for each application to be developed for the mission. This document contains the following information:

- List of classes to be configured into the application
- List of objects to be instantiated from the classes and the mission specific values for the parameters in the objects.
- List of displays and reports to be generated by the application
- Map of which displays and reports are to be configured for each application decision point
- For each display/report, the names of the parameters to be displayed, and display formats.
- Specifications for any new mission/application specific classes, or mission/application specific modifications to a generalized class.

The mission analysis phase is complete when the final versions of the mission application specification documents for each application needed for the mission are completed and the system is operational. The mission analysts deliver the system operations concepts, system requirements, and application specification documents to the application configuration team and the application test team. Updates to the generalized

specifications for the mission are developed by the domain analysis team and delivered to the component engineering team for production. The component engineering team delivers the updated classes to the application configuration team, which configures the updated classes into the mission applications. Figure 8 presents a summary of the mission analysis phase activities and products.

Activities	Products
Define system operations concepts	• Operations Concepts document • Operations Concepts Review (OCR)
Define system requirements	• System Requirements document • System Requirements Review (SRR)
Map system requirements to GSS classes	• Classes to requirements mapping by application • List of modified classes • List of new classes (generalized and/or mission specific)
Define displays and reports	• Displays and report definitions, containing the parameters to be displayed and the display formats
Define application behavior model	• List of displays to be configured at each decision point
Develop Application Specification document	• Application Specification document for each application to be configured for mission support
Develop Specifications for new and modified classes	• Updated GSS Generalized Specification documents (for generalized classes) • Detailed class specifications included in the application specification document (for mission/application specific classes)

Figure 8. Mission Analysis Phase Activities and Products

6. Application Configuration Phase

The application configuration phase begins after the mission analysis team delivers the application specification documents. The phase start also requires that the component engineers have previously populated the reuse asset library with the classes and categories stipulated by the domain analysis team as necessary for configuring the given application.

As the discussion of the previous phases has shown, the GSS process does not fit naturally into a traditional model of sequential non-overlapping software development phases. Instead, a model consisting of overlapping phases, with each phase identified with a major activity, makes more sense. In keeping with this model, it is more natural to include the activity of *fixing* configuration software defects in the same phase as the *writing* of the configuration software. Thus the latter end of the application configuration phase overlaps the application testing phase, as was shown previously in Figure 4.

Guided by the application specification documents, the application configuration team develops the application configuration data files, which include the parameter definition file, application displays files, and the application message file. If the mission analysis team identified requirements for any new *generalized* classes, the component engineering team implements these new classes in the reuse asset library. The application configuration team, however, implements any *mission specific* classes identified by the mission analysts, as well as any additional mission specific configuration code, colloquially referred to as "glueware."

The application may be built in a series of builds. The scheduling of builds will usually need to be coordinated with the release schedules of the User Interface and Executive (UIX) and the Mission Operations Center (MOC) software. It may also have to consider the timing of releases of the reuse asset library and releases of other GSS applications (telemetry simulator, RTADS, Heads Up Display (HUD), telemetry processor (TP), etc.).

The application configuration team certifies its configuration data files and all mission specific configuration code in one-on-one inspections. The code is kept in a controlled library, but the application configuration team does *not* submit SEL Component Origination Forms (COFs) or SEL Change Report Forms (CRFs) for any of this mission specific software. The original motivation for not submitting these forms was that the application configuration process would quickly evolve to a situation where very little glueware would be required. This appears more likely when the generalized classes and categories are implemented in C++ than in Ada 83. The decision not to submit COFs and CRFs for mission specific software continues to be evaluated.

The team processes the application parameter files through the UIX pre-processor in order to create the Codebase database necessary for access by UIX software. Following this, the team develops an informal integration test plan and conducts the integration testing.

After the conclusion of integration testing, the application configurers support the application test team. They begin by training the application test team in the use of the application software. Then they hold the Acceptance Test Readiness Review (ATRR). Following this meeting, the application configurers support the application test team in

setting up and executing the acceptance tests. In particular, the application configurers use the interactive debugger to display specific intermediate program values. When the application testers uncover software defects in the configurers' code and files, the application configurers fix these and rebuild the application's executable image. Defects found in the generalized classes and categories are fixed by the component engineering team. Defects found in the UIX software are fixed by the UIX developers.

The application configurers also develop the user's guide and the system description document during this phase. The application user's guide is "built on" the information contained in the UIX user's guide and the information contained in the application functional specification. The UIX user's guide provides information on how to use the UIX to run an application. The application functional specification defines the mission specific default values for the application input parameters and provides information on the content of the application displays. The application user's guide provides references to the UIX user's guide and the application functional specification where appropriate.

The application system description document is "built on" the GSS functional specifications documents and on the GSS implementation description document. The implementation description document contains the high level design of the UIX based applications. The functional specifications documents contain the class designs for the classes and categories in the reuse asset library. The application system description document only contains application specific design information and appropriate references to the GSS documentation.

During the application configuration phase the application configuration team handles its own software configuration management. Only at the end of the application configuration and application testing phases is the application program delivered to the Flight Dynamics Configuration Management (FDCM) group, which is responsible for software configuration management during operations. See Figure 9 for a recapitulation of the application configuration phase activities and products.

Activities	Products
Develop application configuration data files Develop mission specific classes and mission specific configuration code	• Parameter definition file • Application displays file • Application message file • Mission specific classes • Mission specific configuration code (“glueware”)
Inspect (one-on-one) application data files and mission specific classes and configuration code	• Certified data files and code
Process configuration data files through the UIX pre-processor	• Codebase database for access by UIX software
Develop informal integration test plan and integration test the application	• Informal integration test plan • Integration tested application program and support files
Support the application test team • Train application test team in use of application • Hold the ATRR • Help set up and execute acceptance test • Use interactive debugger to display specific intermediate program values	• Acceptance Test Readiness Review (ATRR) materials • Debugger output
Fix defects found in configuration data files and mission specific code	• Acceptance tested application
Write user’s guide and system description documents	• User’s guide • System description document
Deliver application program to FDCM	• Configured application software

Figure 9. Application Configuration Phase Activities and Products

7. Application Test Phase

The application testing phase coincides with the latter portion of the application configuration phase. During the testing phase the application testing team conducts the acceptance testing of the configured application.

Prior to the ATRR, the application testers write the application test plan and prepare acceptance test input data. They also receive training from the application configurers in the use of the application software. The testers then participate in the ATRR

Following this meeting, the test team conducts the acceptance testing, assisted by the application configuration team as described in the previous section. Afterwards the application test team evaluates the test results. They document test results in detailed reports and test item checklists. They document software failures in problem reports and help determine the source of failures. After the software defects are fixed (by application configurers, component engineers, or UIX developers, as appropriate), the application testing team retests the application. Finally the test team certifies the application for delivery to mission support for operational use. The activities and products of the application testing phase are summarized in Figure 10.

Activities	Products
Develop application acceptance test plan (draft)	• Draft acceptance test plan
Prepare acceptance test input data	• Acceptance test input data
Attend ATRR	• Acceptance Test Readiness Review (ATRR) materials
Revise acceptance test plan	• Updated acceptance test plan
Conduct acceptance test of configured application and evaluate test results • Generate detailed reports • Generate test item checklists • Document software failures in problem reports • Work with mission analysts and application configurers to determine probable source of a failed test item • Retest software fixes	• Acceptance test detailed reports • Acceptance test item checklists • Software problem reports
Certify application for delivery to mission support for operational use	• Validated software application

Figure 10. Application Testing Phase Activities and Products